

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of :
Maiko TARUKI et al. :
Serial No. NEW : **Attn: APPLICATION BRANCH**
Filed September 23, 2003 : Attorney Docket No. 2003_1310A

**SIMULATOR FOR SOFTWARE
DEVELOPMENT AND RECORDING
MEDIUM HAVING SIMULATION
PROGRAM RECORDED THEREIN**

THE COMMISSIONER IS AUTHORIZED
TO CHARGE ANY DEFICIENCY IN THE
FEES FOR THIS PAPER TO DEPOSIT
ACCOUNT NO. 23-0975

CLAIM OF PRIORITY UNDER 35 USC 119

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

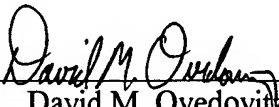
Sir:

Applicants in the above-entitled application hereby claim the date of priority under the International Convention of Japanese Patent Application No. 2002-280827, filed September 26, 2002, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Maiko TARUKI et al.

By 
David M. Ovedovitz
Registration No. 45,336
Attorney for Applicants

DMO/jmj
Washington, D.C. 20006-1021
Telephone (202) 721-8200
Facsimile (202) 721-8250
September 23, 2003

日 本 国 特 許 庁

JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application:

2002年 9月26日

出 願 番 号

Application Number:

特願2002-280827

[ST.10/C]:

[JP 2002-280827]

出 願 人

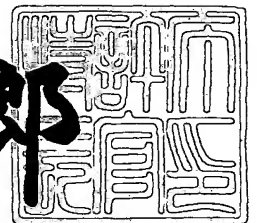
Applicant(s):

松下電器産業株式会社

2003年 4月25日

特 許 庁 長 官
Commissioner,
Japan Patent Office

太田 信一郎



出証番号 出証特2003-3030363

【書類名】 特許願

【整理番号】 2838240102

【提出日】 平成14年 9月26日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/45

【発明者】

 【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

 【氏名】 樽木 麻衣子

【発明者】

 【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

 【氏名】 中村 剛

【発明者】

 【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

 【氏名】 近藤 孝宏

【特許出願人】

 【識別番号】 000005821

 【氏名又は名称】 松下電器産業株式会社

【代理人】

 【識別番号】 100097179

 【弁理士】

 【氏名又は名称】 平野 一幸

【手数料の表示】

 【予納台帳番号】 058698

 【納付金額】 21,000円

【提出物件の目録】

 【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 0013529

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 シミュレータ、そのシミュレータをコンピュータ読み取り可能に記録した記録媒体

【特許請求の範囲】

【請求項 1】 高級言語で記述されたソースコードをコンパイルするコンパイラと、

前記ホストプロセッサとは異なる、ターゲットプロセッサのハードウェア構成要素を、高級言語でモデル化する関数又は手続きと、

この関数又は手続きを利用して、ターゲットプロセッサの命令に対応して高級言語で定義された関数又は手続きとを含む

ライブラリとを備え、

前記ソースコードは、このライブラリを使用して記述される、シミュレータ。

【請求項 2】 前記ハードウェア構成要素には、ターゲットプロセッサの演算器と、ターゲットプロセッサのメモリコントローラと、ターゲットプロセッサのレジスタとが含まれる、請求項 1 記載のシミュレータ。

【請求項 3】 前記ライブラリは、

ターゲットプロセッサのハードウェア構成要素を、高級言語でモデル化する関数又は手続きを定義するハードウェアモデルライブラリと、

前記ハードウェアモデルライブラリにおける、関数又は手続きを利用して、ターゲットプロセッサの命令に対応する関数又は手続きを、高級言語で定義する命令セットライブラリと

を備える、請求項 1 から 2 記載のシミュレータ。

【請求項 4】 前記ターゲットプロセッサの命令には、ADD 命令、SUB 命令、AND 命令、OR 命令、LD 命令、ST 命令、SET 命令及びMOV 命令が含まれる、請求項 1 から 3 記載のシミュレータ。

【請求項 5】 前記ライブラリには、ターゲットプロセッサにおける実行サイクル数、消費電力のうち、一方又は双方を計算する関数又は手続きが含まれている、請求項 1 から 4 記載のシミュレータ。

【請求項 6】ターゲットプロセッサにおける、実行サイクル数、消費電力のうち、一方又は双方は、変更可能となっている、請求項 1 から 4 記載のシミュレータ。

【請求項 7】前記ライブラリには、ターゲットプロセッサにおけるコードサイズを計算する関数又は手続きが含まれている、請求項 1 から 6 記載のシミュレータ。

【請求項 8】オブジェクトコードをフェッチする命令フェッチ手段と、フェッチされたオブジェクトコードをデコードする命令デコード手段と、デコードされた命令をホストプロセッサ上で実行する実行手段と、高級言語で記述されたソースコードを読み込んでオブジェクトコードを生成する翻訳装置と、

前記ホストプロセッサとは異なる、ターゲットプロセッサのハードウェア構成要素を、高級言語でモデル化する関数又は手続きと、

この関数又は手続きを利用して、ターゲットプロセッサの命令に対応して高級言語で定義された関数又は手続きとを含むライブラリとを備え、

前記ソースコードは、このライブラリを使用して記述される、シミュレータ。

【請求項 9】請求項 1 から 8 記載のシミュレータが、コンピュータ読み取り可能に記録された記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、ターゲットプロセッサ向け組み込みソフトウェアの開発に適した、シミュレータ及びその関連技術に関するものである。

【0002】

【従来の技術】

ターゲットプロセッサ向け組み込みソフトウェアを開発する際、一般に、ターゲットプロセッサとは互換性のない、ホストプロセッサ上で動作するソフトウェア開発環境が用いられる。

【0003】

ここで、本明細書において、「ホストプロセッサ」とは、このソフトウェア開発環境を動作させ、開発と、開発の結果物であるソフトウェアの検証とに、使用するプロセッサをいう。

【0004】

また、「ターゲットプロセッサ」とは、ホストプロセッサとは異なるプロセッサであり、この開発の結果物であるソフトウェアを実行するプロセッサをいう。

【0005】

そして、ホストプロセッサとターゲットプロセッサとは、ソフトウェア互換性がない。また、このような、結果物としてのソフトウェアは、専らターゲットプロセッサ上でのみ正常に動作し、ホストプロセッサ上では正常に動作しない。

【0006】

また、本明細書にいう「シミュレータ」が生成するソフトウェアは、上記結果物であるソフトウェアを模したソフトウェアであるが、ターゲットプロセッサ上で動作するものではなく、ホストプロセッサ上で動作するものである。

【0007】

このように、ホストプロセッサのソフトウェア開発環境を用いて、ホストプロセッサとは異なるターゲットプロセッサ向けのソフトウェアを開発する場合、次に述べるような問題点がある。

【0008】

ここで近年、このようなソフトウェアを開発する場合であっても、ソフトウェアが大規模になるに伴い、アセンブラ言語主体による開発から、C/C++等の高級言語による開発へ、シフトしている。

【0009】

これは、高級言語を用いたソフトウェア開発を行えば、データの保持、転送、演算等の処理を、ターゲットプロセッサのアセンブラレベルの命令やレジスタ、メモリ等のリソースに依存しない形式で記述でき、可読性、汎用性、開発効率に優れているためである。

【0010】

ここで特に、組み込み向けソフトウェア等では、システムのパフォーマンスを最大限引き出すため、プロセッサの能力を極限まで高める、最適化が求められる。

【0011】

しかしながら、高級言語によるソフトウェア開発では、コンパイラ性能の問題から高級言語からアセンブラコードに変換する際に、冗長なコードが生成され、ソフトウェアのコードサイズや実行速度に影響を及ぼす場合がある。

【0012】

したがって現在でも、信号処理などの処理負荷が大きい部分に関しては、高級言語による開発に加え、ターゲットプロセッサのアセンブラ言語による開発が行われている。ところが、これでは、ソフトウェア開発の効率が落ちてしまう。

【特許文献1】

特開2000-330821号公報（第4-8頁）

【0013】

【発明が解決しようとする課題】

組み込み向けソフトウェア開発では、ソフトウェア開発の大規模化・複雑化に伴い、その開発工数が増大する傾向にある。加えて、検証工数、シミュレーション時間の増大も著しい。

【0014】

そして、ターゲットプロセッサの完成前に、ターゲットプロセッサを入手できないまま、ソフトウェア開発に着手しなければならないケースが多い。

【0015】

また、ソフトウェア開発環境が十分に整備されていない段階でも、プログラムのパフォーマンス解析を行いたいものである。しかしながら、LSIの大規模化・複雑化に、シミュレータの速度が対応できておらず、短期間で高精度な情報を得ることは難しい。

【0016】

さらに、従来技術では、コンパイラやシミュレータなどのソフトウェア開発環境の整備にかかる工数が多く、この開発環境を用いた、目的のソフトウェア開発

の着手が遅れがちとなる。

【0017】

そこで本発明は、ソフトウェアを効率良く開発できる、コンパイラ型及びインタープリタ型の2種類の、シミュレータを提供することを目的とする。より詳しくは、本発明は、アセンブラレベルの最適化（コードサイズ、実行速度の両面で）も含め、全て高級言語で開発された1つのソフトウェアを、これら2種類のシミュレータで、共用して、シミュレーションできる、シミュレータを提供するものである。

【0018】

【課題を解決するための手段】

請求項1記載のシミュレータは、高級言語で記述されたソースコードをコンパイルするコンパイラと、ホストプロセッサとは異なる、ターゲットプロセッサのハードウェア構成要素を、高級言語でモデル化する関数又は手続きと、この関数又は手続きを利用して、ターゲットプロセッサの命令に対応して高級言語で定義された関数又は手続きを含むライブラリとを備え、ソースコードは、このライブラリを使用して記述される。

【0019】

この構成において、ターゲットプロセッサのハードウェア構成要素を、ライブラリにおいて関数又は手続きにより、モデル化し、ソースコードをこのライブラリを利用して記述するようにしている。これにより、コンパイラ型のシミュレータを構成できる。

【0020】

コンパイラ型のシミュレータが出力するオブジェクトコードは、命令フェッチや命令デコードなどのプロセスを経ずに実行できるため、実行速度が高速である。

【0021】

また、ソースコードが全て高級言語で記述されるため、移植性や可読性に優れた、開発を行えるし、アセンブラレベルの最適化も含め、全て高級言語でのソフトウェア開発が可能となる。

【0022】

しかも、後述するように、このソースコードは、インタプリタ型のシミュレータにも共用でき、コンパイラ型、インタプリタ型の2種類のシミュレータを開発するにあたり、部品を共通化でき、開発効率が向上する。

【0023】

請求項2記載のシミュレータでは、ハードウェア構成要素には、ターゲットプロセッサの演算器と、ターゲットプロセッサのメモリコントローラと、ターゲットプロセッサのレジスタとが含まれる。

【0024】

この構成により、ターゲットプロセッサの主要な構成要素である、演算器、メモリコントローラの振る舞いを、モデル化により、表現できる。

【0025】

請求項3記載のシミュレータでは、ライブラリは、ターゲットプロセッサのハードウェア構成要素を、高級言語でモデル化する関数又は手続きを定義するハードウェアモデルライブラリと、ハードウェアモデルライブラリにおける、関数又は手続きを利用して、ターゲットプロセッサの命令に対応する関数又は手続きを、高級言語で定義する命令セットライブラリとを備える。

【0026】

この構成において、ライブラリを、ハードウェアモデルライブラリと、命令セットライブラリとに分けているため、ライブラリの取り扱いが容易になる。即ち、ハードウェアモデルライブラリを、後述するインタプリタ型のシミュレータと共有すると共に、コンパイラ型のシミュレータには、命令セットライブラリが追加されることになる。

【0027】

言い換えれば、この構成に係るコンパイラ型のシミュレータは、ハードウェアモデルライブラリに、命令セットライブラリを追加するだけで実現可能となり、翻訳装置などが不要であるため、インタプリタ型のシミュレータに比べ、簡易に構成できる。このため、シミュレータの開発工数が減少し、早期提供が可能となる。

【0028】

請求項4記載のシミュレータでは、ターゲットプロセッサの命令には、ADD命令、SUB命令、AND命令、OR命令、LD命令、ST命令、SET命令及びMOV命令が含まれる。

【0029】

この構成により、ターゲットプロセッサの主要な命令をカバーできる。

【0030】

請求項5のシミュレータでは、ライブラリには、ターゲットプロセッサにおける実行サイクル数、消費電力のうち、一方又は双方を計算する関数又は手続きが含まれている。

【0031】

この構成により、ライブラリに含まれる、サイクル数、消費電力等を計算する関数又は手続きによる、計算結果を出力させれば、シミュレーション終了時にアセンブラプログラムの実行サイクル数、消費電力等を得ることができ、コンパイラ型のシミュレータにおいて、パフォーマンス解析が可能になる。

【0032】

請求項6記載のシミュレータでは、ターゲットプロセッサにおける、実行サイクル数、消費電力のうち、一方又は双方は、変更可能となっている。

【0033】

この構成により、実行サイクル数、消費電力等を意図的に変更することにより、様々な条件下での、シミュレーションを行える。

【0034】

請求項7記載のシミュレータでは、ライブラリには、ターゲットプロセッサにおけるコードサイズを計算する関数又は手続きが含まれている。

【0035】

この構成により、コードサイズによる最適化や、パフォーマンス解析を行える。

【0036】

請求項8記載のシミュレータは、オブジェクトコードをフェッチする命令フェ

ッチ手段と、フェッチされたオブジェクトコードをデコードする命令デコード手段と、デコードされた命令をホストプロセッサ上で実行する実行手段と、高級言語で記述されたソースコードを読み込んでオブジェクトコードを生成する翻訳装置と、ホストプロセッサとは異なる、ターゲットプロセッサのハードウェア構成要素を、高級言語でモデル化する関数又は手続きと、この関数又は手続きを利用して、ターゲットプロセッサの命令に対応して高級言語で定義された関数又は手続きとを含むライブラリとを備え、ソースコードは、このライブラリを使用して記述される。

【 0 0 3 7 】

この構成により、インタプリタ型のシミュレータを構成できる。しかも、このインタプリタ型のシミュレータのソースコードは、上述したコンパイラ型のシミュレータのそれと、共用でき、コンパイラ型、インタプリタ型、都合 2 種類のシミュレータ上で動作させるプログラムを一本化でき、プログラムの開発効率を向上できる。

【 0 0 3 8 】

【発明の実施の形態】

以下、図面を参照しながら、本発明の実施の形態を説明する。なお、具体的な説明に先立ち、本明細書にいう高級言語及びその関数又は手続きについて、簡単に述べる。

【 0 0 3 9 】

以下の各形態では、高級言語として、C 言語を用いる。C 言語の仕様上、戻り値のない (void 型) の処理を、” 手続き ” とは呼ばず、” 関数 ” と呼ぶ習慣がある。しかしながら、他の高級言語、例えば P a s c a l では、戻り値のない手続き (p r o c e d u r e) と、戻り値を持つ関数 (f u n c t i o n) とを、厳格に区別している。

【 0 0 4 0 】

ここで、本明細書にいう高級言語は、C 言語に限定されず、その上位の C ++ 言語、P a s c a l などを含む。

【 0 0 4 1 】

したがって、本明細書にいうライブラリでは、ターゲットプロセッサのハードウェア構成要素は、”関数又は手続きにより”モデル化されており、この関数又は手続きを利用し、ターゲットプロセッサの命令に対応する、”関数又は手続き”は、高級言語で定義されているということになる。

【0042】

(実施の形態1)

次に、本形態の具体的構成を説明する。さて、図1は、本発明の実施の形態1におけるソフトウェア開発環境のブロック図である。大きく分けて、この開発環境は、コンパイラ型のシミュレータ102と、インタプリタ型のシミュレータ108との、2種類のシミュレータから構成される。

【0043】

勿論、本発明のシミュレータは、コンパイラ型あるいはインタプリタ型のシミュレータを、単独に構成することもできるし、このように、単独に構成したシミュレータも、本発明に包含される。

【0044】

図1に示しているように、これら、コンパイラ型のシミュレータ102、インタプリタ型のシミュレータ108は、いずれも同一のハードウェアモデルライブラリ101を利用する。

【0045】

ハードウェアモデルライブラリ101は、後にソースコード例を示して詳述するように、ターゲットプロセッサのハードウェア構成要素（演算器、メモリコントローラ、レジスタなど）を、C言語でモデル化した、関数、変数等を定義する。

【0046】

また、命令セットライブラリ105は、ハードウェアモデルライブラリ101における、関数等を利用して、ターゲットプロセッサの命令に対応する関数等を、C言語で定義する。

【0047】

本形態のライブラリは、ハードウェアモデルライブラリ101と、命令セット

ライブラリ105とからなる。

【0048】

また、このライブラリには、コンパイラ型のシミュレータ102によるソフトウェアのパフォーマンス解析を可能にするため、サイクル数や消費電力、リソースの使用状況などを計算する関数が設けられている。

【0049】

このうち、ハードウェアモデルライブラリ101は、コンパイラ型のシミュレータ102、インタプリタ型のシミュレータ108の、2種類のソフトウェアシミュレータに共通に適用される。

【0050】

すなわち、ターゲットプロセッサ内のハードウェア機能を実現する関数群を提供するライブラリを用意し、そのライブラリ関数、変数を、コンパイラ型、インタプリタ型のシミュレータの構成要素として利用する。

【0051】

次に、コンパイラ型のシミュレータ102の概要について説明する。ソースコード103は、ハードウェアモデルライブラリ101、命令セットライブラリ105の定義を利用して記述された、C言語のプログラムである。このソースコード103は、インタプリタ型のシミュレータ108のソースコードでもある。

【0052】

コンパイラ104は、ホストプロセッサ上で動作する、C言語のコンパイラである。このコンパイラ104は、通常のCコンパイラであれば十分である。そして、コンパイラ104は、ソースコード103をコンパイルして、オブジェクトファイル106を出力する。

【0053】

命令セットライブラリ105は、後述するこのライブラリのソースコードを既にコンパイルしてあり、オブジェクトファイルの形態になっている。

【0054】

リンカ100は、オブジェクトファイル106と、命令セットライブラリ105（オブジェクトファイル）とを、リンクして、機械語のオブジェクトコード1

07を出力する。このオブジェクトコード107は、ホストプロセッサ上で動作するものであり、ターゲットプロセッサ上では、一般には、動作しない。

【0055】

次に、インタプリタ型のシミュレータ108の概要について、説明する。まず、翻訳装置112は、C言語で記述したプログラムを、ターゲットプロセッサ用のオブジェクトコード113に翻訳する。

【0056】

オブジェクトコード113は、実行時に、命令フェッチ手段109により、1命令ずつ読み出され、命令デコード手段110によって解読され、実行手段111により実行される。

【0057】

次に、図2を用いて、本形態で想定するターゲットプロセッサの構成を説明する。図2は、本発明の実施の形態1で使用するターゲットプロセッサのブロック図である。勿論、図示しているターゲットプロセッサは例示にすぎず、他の形態を採るターゲットプロセッサについても、本発明は、同様に適用できる。

【0058】

図2に示すターゲットプロセッサは、次の要素を有する。まず、命令レジスタ(IR)201は、実行中の命令を保持するレジスタである。

【0059】

プログラムカウンタ(PC)205は、現在実行中の命令のアドレスを保持するレジスタである。

【0060】

レジスタ群206は、後述するメモリから読み出したデータや演算結果を保持するものであり、本形態では、4個の16ビットのレジスタで構成される。これらのレジスタを、それぞれ、レジスタR0(209)、レジスタR1(210)、レジスタR2(211)、レジスタR3(213)と呼ぶ。

【0061】

図4(a)に示すように、演算器(ALU)207は、加減算(ADD、SUB)、論理演算(AND、OR)計4種類の16ビットバイナリ演算を実行する

ユニットであり、2つのデータ入力（IN1、IN2）、1つのデータ出力（OUT）、ならびに1つの制御信号（CTL）を持つ。

【0062】

また、図4（b）に示すように、メモリコントローラ208は、データメモリA（203）、データメモリB（204）へのアクセスを制御するユニットであり、アドレス値によりアクセスするデータメモリを選択する。

【0063】

なお、図4（b）において、AD信号は、アクセスするメモリへのアドレス値を表す。DB信号は、ポインタである。メモリコントローラ208は、リード時に、メモリ変数から読み出したデータを、DB信号が指すポインタへデータを格納し、ライト時に、DB信号が指すポインタ内のデータを読み出しメモリ変数へ格納する。RW信号は、リード／ライトを表す制御信号である。

【0064】

また、図1において、以上のターゲットプロセッサには、バス200を介して、次のメモリが接続される。

【0065】

命令メモリ（IMEM）202は、プログラムを格納するためのメモリであり、データメモリA（203）、データメモリB（204）は、ターゲットプロセッサが、演算に使用するデータを格納するためのメモリである。

【0066】

次に、図3を用いて、ターゲットプロセッサの命令セットを説明する。図3（a）は、本発明の実施の形態1におけるターゲットプロセッサの命令セットの説明図である。

【0067】

図3（a）に示すように、このターゲットプロセッサは、ADD命令、SUB命令、AND命令、OR命令、LD命令、ST命令、SET命令、MOV命令の、8種類の命令セットを持つ。

【0068】

また、ターゲットプロセッサは、レジスタの指定について、図3（b）に示す

規則に従う。

【0069】

ADD命令では、ターゲットプロセッサは、指定された2つのオペランドレジスタ内に格納されているデータに対して加算を行い、結果を第1オペランドレジスタへ格納する。

【0070】

SUB命令では、ターゲットプロセッサは、指定された2つのオペランドレジスタ内に格納されているデータに対して減算を行い、結果を第1オペランドレジスタへ格納する。

【0071】

AND命令では、ターゲットプロセッサは、指定された2つのオペランドレジスタ内に格納されているデータに対し、論理積を計算し、結果を第1オペランドレジスタへ格納する。

【0072】

OR命令では、ターゲットプロセッサは、指定された2つのオペランドレジスタ内に格納されているデータに対し、論理和を計算し、結果を第1オペランドレジスタへ格納する。

【0073】

LD命令では、ターゲットプロセッサは、第2オペランドレジスタで指定したレジスタに格納されているデータを、データメモリA（203）またはデータメモリB（204）のアドレスとして、データメモリ内のデータを第1オペランドレジスタへ格納する。

【0074】

ST命令では、ターゲットプロセッサは、第2オペランドレジスタに格納されているデータを、データメモリA（203）またはデータメモリB（204）のアドレスに、第1オペランドレジスタ内のデータを格納する。

【0075】

SET命令では、ターゲットプロセッサは、第2オペランドで指定した即値を第1オペランドレジスタへ格納する。

【 0 0 7 6 】

MOV 命令では、ターゲットプロセッサは、第 2 オペランドレジスタに格納されているデータを第 1 オペランドレジスタへ格納する。

【 0 0 7 7 】

次に、図 5、図 6 を用いて、図 2 に示すターゲットプロセッサの各要素をモデル化した、ハードウェアモデルライブラリ 1 0 1 の構成例について述べる。

【 0 0 7 8 】

ここで、図 5 は、本発明の実施の形態 1 におけるハードウェアモデルライブラリのヘッダファイルの例示図、図 6 は、本発明の実施の形態 1 におけるハードウェアモデルライブラリの実装ファイルの例示図である。

【 0 0 7 9 】

なお、以下図 2、図 3 において名付けた要素に対応する、プログラムの要素には、同じ名前を付与する。例えば、図 2 の命令メモリ (IMEM) 2 0 2 に対応する、プログラムの要素には、IMEM という名前を付与する。

【 0 0 8 0 】

さて、ここでは、図 5 のようなヘッダファイルで、変数や関数の宣言を行い、その内容は、図 6 の実装ファイルで記述している。しかしながら、ヘッダファイルと実装ファイルとに分離しなくとも良く、1 つのファイルで全て記述してしまってもかまわない。

【 0 0 8 1 】

図 4 及び図 5 に示すように、配列 IMEM、DMEMA、DMEMB は、それぞれ、命令メモリ 2 0 2、データメモリ A (2 0 3)、データメモリ B (2 0 4) を表し、それぞれのサイズ分の要素を持つ。勿論、メモリをモデル化するデータ構造は、配列でなくとも良く、リストなど、他の周知の構造を使用して差し支えない。

【 0 0 8 2 】

変数 IR、PC、R 0、R 1、R 2、R 3 は、それぞれ、命令レジスタ (IR) 2 0 1、プログラムカウンタ (PC) 2 0 5、レジスタ R 0 (2 0 9)、レジスタ R 1 (2 1 0)、レジスタ R 2 (2 1 1)、レジスタ R 3 (2 1 3) を表し

、16ビットの整数型である `short` 型の変数で宣言する。

【0083】

演算器 (ALU) 207は、図4 (a) に示すように、変数 `IN1`、`IN2`、`OUT`、`CTL`の4つの引数を持つ、ALU関数で表現する。

【0084】

ここで、変数 `IN1`、`IN2`は2つのデータ入力を表し、`short`型とする。変数 `OUT`は、データ出力を表し、`short`型のポインタとする。変数 `CTL`は、制御信号を表し、`int`型とする。なお、ALU関数410は、ハードウェアと同じ演算精度となるようC言語で記述してある。

【0085】

メモリコントローラ (MEMC) 208は、図4 (b) に示すように、変数 `AD`、`DB`、`RW`の3つの引数を持つMEMC関数で、表現する。

【0086】

ここで、変数 `AD`は、アクセスするメモリへのアドレス値を表す。変数 `DB`は、ポインタであり、MEMC関数は、リード時に、メモリ変数から読み出したデータを、変数 `DB`の指すポインタへデータを格納し、ライト時に、変数 `DB`の指すポインタ内のデータを読み出しメモリ変数へ格納する。なお、変数 `RW`は、リード／ライトを表す制御信号である。

【0087】

本形態では、以上の9個の変数 (配列も含めている) `IMEM`、`DMEMA`、`DMEMB`、`IR`、`PC`、`R0`、`R1`、`R2`、`R3`と、2つのライブラリ関数 (ALU関数、MEMC関数) を含めて、ハードウェアモデルライブラリ101を構成している。

【0088】

次に、図7を用いて、命令セットライブラリ105の実装例について、説明する。但し、オブジェクトファイルにする前の、ソースコードレベルで述べる。

【0089】

図3 (a) を参照しながら上述したように、ターゲットプロセッサは、`ADD`命令、`SUB`命令、`AND`命令、`OR`命令、`LD`命令、`ST`命令、`SET`命令、

MOV命令の、8種類の命令セットを持っている。そこで、本形態の命令セットライブラリ105では、これらの命令に、一対一に対応する次の関数を用意している。

【0090】

図7に示しているように、ADD関数、SUB関数、AND関数、OR関数は、それぞれ、2つの引数RS1、RS2を持つ関数として定義され、ハードウェアモデルライブラリ101のライブラリ関数であるALU関数410を、呼び出すことにより、演算が実施されるようになっている。

【0091】

ここで、引数RS1、RS2は、それぞれ命令セットの第1、第2オペランドレジスタに対応する。

【0092】

また、LD関数、ST関数は、2つの引数RS1、RS2を持つ関数として定義され、ハードウェアモデルライブラリ101のライブラリ関数であるMEMC関数415を、呼び出すことにより、メモリのコントロールが実施されるようになっている。

【0093】

ここで、引数RS1、RS2は、それぞれ命令セットの第1、第2オペランドレジスタに対応する。

【0094】

SET関数は、2つの引数RD、IMDを持つ関数として定義され、引数IMDの値を、*RS1に代入する。

【0095】

MOV関数は、2つの引数RD、RSを持つ関数として定義され、引数RS2の値を、*RS1に代入する。

【0096】

本形態の命令セットライブラリ105には、以上の8つの命令に対応した関数を用意されている。そして、図7の実装ファイルを、プログラム用のソースコードでインクルードしやすいように、図8(a)に示すような、ヘッダファイルを

用意しておく。

【0097】

次に、図8を用いて、図2に示すソースコード103の例について述べる。まず、図8のソースコード103では、冒頭で、図8(a)に示したヘッダファイルをインクルードしており、以下、図7で実装した、命令セットライブラリ105の各関数を利用できる。

【0098】

そして、main関数内において、SET関数、MOV関数等呼び出して、所望の処理を行うようにしている。なお、このソースコード103には、命令セットライブラリ105に含まれる関数のみしか記述できないものではなく、通常のC言語で利用できる関数等を記述することができる。

【0099】

また、このようにすると、ソースコード103→命令セットライブラリ105→ハードウェアモデルライブラリ101の順に、呼び出しが行われることになる。

【0100】

このソースコード103を、パーソナルコンピュータやワークステーションなどのホストプロセッサ上で、コンパイラ104を用いて、コンパイルし実行することにより、ターゲットプロセッサの動作をシミュレーションすることができる。

【0101】

次に、図9を用いて、図2に示す、インタプリタ型のシミュレータ108について説明を追加する。まず、翻訳装置112について、説明する。

【0102】

図9(a)には、本形態のインタプリタ型のシミュレータ108による処理の流れを示してあり、図9(b)には、それに対応して、アセンブラ902で処理を行う場合の流れを示している。

【0103】

さて、図9(a)に示すように、本形態の翻訳装置112は、ソースコード1

03（コンパイラ型のシミュレータ102と同じもの）を読み込んで、オブジェクトコード113を出力する必要がある。

【0104】

一方、ソースコード103（C言語）に対応する、アセンブラプログラム901がある時、アセンブラ902が、このアセンブラプログラム901をアセンブルして、オブジェクトコード903を出力する。

【0105】

ここで、これらのオブジェクトコード113、903が一致すれば、目的を達成できるわけである。

【0106】

即ち、この翻訳装置112を簡単に構成するならば、翻訳装置112は、
（処理1）ソースコード103を入力し、これを、アセンブラプログラム901に置換する、

（処理2）翻訳装置112は、アセンブラ902相当の機能を内蔵し、置換したアセンブラプログラム901を、アセンブルする、
という、2つの処理を行うようにすればよい。

【0107】

ここで、（処理1）は、*、?をワイルドカードとして、
（規則1）「#include *」という行を削除する、
（規則2）「main()*{」を「main:」に置換する、
（規則3）「SET(&*, ?);」を「SET *, ?」に置換する、
...（規則n）という、置換規則に従えば、単なる文字列の置換により、対応できる。また、（処理2）もアセンブラを流用すればよいから、簡単に対応できる。

【0108】

但し、翻訳装置112は、（処理1）と（処理2）とを分離して実施する必要はなく、一度に処理を完了するようにしても良い。

【0109】

次に、図10を用いて、図2の命令フェッチ手段109、命令でコード手段1

10、実行手段111について説明する。

【0110】

図10は、本発明の実施の形態1におけるインタプリタ型のシミュレータの実装ファイルの例示図である。

【0111】

まず、冒頭で、ハードウェアモデルライブラリ101のヘッダファイルをインクルードしている。そして、状態を示す変数 `state` の型を、「Fetch」、「Decode」、「Exec」の3つだけ、列挙して宣言している。もちろん、変数 `state` が「Fetch」の場合は、命令フェッチ手段109が動作すべきことを示し、同様に、変数 `state` が「Decode」、「Exec」の場合は、それぞれ、命令デコード手段110、実行手段111が動作すべきことを示す。また、サイクル数を記憶する変数 `cycle` を、`int` 型で宣言し導入している。

【0112】

そして、`main` 関数において、はじめに変数 `cycle` を「0」とし、変数 `state` を「Fetch」にしてから処理を開始する。

【0113】

次の `while` 分において、`exec` 関数を呼び出し、最後に変数 `cycle` の値を、標準出力している。これにより、実行サイクル数を計測できる。

【0114】

`exec` 関数では、呼び出される毎に、変数 `cycle` が「+1」され、以下、`switch` 文において、変数 `state` によって、分岐して処理が実行される。

【0115】

変数 `state` が「Fetch」の場合は、命令フェッチ手段109に相当する処理 (`case Fetch`: 以下 `case Decode`: の直前まで) が実行される。

【0116】

変数 `state` が「Decode」の場合は、命令デコード手段110に相当

する処理（case Decode：以下case Exec：の直前まで）が実行される。

【0117】

変数stateが「Exec」の場合は、実行手段111に相当する処理（case Decode：以下）が実行される。

【0118】

ここで、以上の各処理において、配列IMEMやALU関数など、ハードウェアモデルライブラリ101で定義された、変数、配列、関数などが利用されている点に注目されたい。

【0119】

<実施の形態1の効果>

以上のように、ターゲットプロセッサの構成要素を単位として、機能を関数などでモデル化したライブラリを要し、これを利用することにより、コンパイラ型102、インタプリタ型のシミュレータ108の開発効率が向上する。

【0120】

命令セットライブラリ105は、単なるソースコードを用意し、それをコンパイルしておくだけで良いから、インタプリタ型のシミュレータ108に比べ構成が容易であり、開発工数が減少し、インタプリタ型108に比べ早期提供が可能となる。

【0121】

また、命令セットライブラリ105の関数を用いて記述した、ソースコード103から、アセンブラコードで記述したコードをアセンブルして得られるオブジェクトコードと同じオブジェクトコードを生成する翻訳装置112を組み入れることで、2種類のシミュレータ上で動作させるプログラムを一本化できる。

【0122】

（実施の形態2）

以下、実施の形態1との相違点について説明する。

まず、図11に示すように、ハードウェアモデルライブラリ101において、サイクル数ならびに消費電力を表す変数cycle、powerを追加する。

【0123】

また、ALU関数において、実際のハードウェアで規定されているALU演算にかかるサイクル数と消費電力を、それぞれ変数`cycle`、`power`に加える処理を追加している。

【0124】

MEMC関数において、実際のハードウェアで規定されているメモリアクセスに要するサイクル数ならびに消費電力をそれぞれ、変数`cycle`、`power`に加える処理を追加している。

【0125】

次に、図12に示すように、命令セットライブラリ105において、SET関数、MOV関数につき、命令実行に要するサイクル数と消費電力を計算するために、変数`cycle`、`power`を加算する処理を追加している。

【0126】

さらに、図13に示すように、ソースコード103において、変数`cycle`、`power`の値を、`printf`関数を用いて標準出力する行を追加している。勿論、標準出力せず、変数`cycle`、`power`の値を、ファイルに書き出すなど種々変更して差し支えない。

【0127】

これにより、プログラムの実行に要したサイクル数と消費電力に関する情報を得ることができる。

【0128】

図示していないが、実施の形態2における、インタプリタ型のシミュレータ108では、以上と同様に、図10の`main`関数において、変数`power`の値を`printf`関数を用いて標準出力する機能等を追加すると良い。

【0129】

これにより、インタプリタ型のシミュレータ108においても、プログラムの実行に要した消費電力に関する情報を得ることができる。

【0130】

<実施の形態2の効果>

実施の形態1の効果に加え、ライブラリにサイクル数ならびに消費電力を計算する処理を組み込み、対応する変数を出力させることで、シミュレーション終了時にアセンブラプログラムの実行サイクル数ならびに消費電力を得ることができ、コンパイラ型、インタプリタ型の、2種類のシミュレータにおいてパフォーマンス解析が可能になる。

【0131】

（実施の形態3）

実施の形態3においては、実施の形態2との相違点のみを説明する。

まず、図14に示すように、命令セットライブラリ105において、サイクル数を表す変数 `cycle`、消費電力を表す変数 `power`、コードサイズを表す変数 `code` を、追加する。

【0132】

また、本形態では、図15（a）に表でまとめているように、ADD命令、SUB命令など、命令毎にインデックスをふり、変数 `cycle` の増分、変数 `power` の増分を定め、これらの増分を、配列 `cycle_tbl[]`、配列 `power_tbl[]` に格納する。勿論、配列でない他の構造でこれらの増分を格納しても良い。

【0133】

そして、図14に示すように、命令セットライブラリ105において、ADD関数、SUB関数、LD関数、ST関数、MOV関数において、これらの変数 `cycle`、`power`、`code` を加算する処理を追加する。ここで、変数 `cycle`、`power` を計算する際は、各命令実行時に要するデータが格納されている配列 `cycle_tbl[]`、`power_tbl[]` の値を利用する。

【0134】

なお、図15（b）のソースコード例に示すように、配列 `cycle_tbl[]`、`power_tbl[]` の値のデータを、ファイル（図示の例では“`table`”）に格納しておき、初期化時に、ファイルから配列 `cycle_tbl[]`、`power_tbl[]` にロードするようにすると良い。

【0135】

なお、この `init` 関数は、シミュレーション開始前に予め実行する初期化関数である。

【0136】

＜実施の形態3の効果＞

実施の形態2の効果に加え、命令実行に要するサイクル数、消費電力に関する情報を、外部（本形態ではファイル）から与える仕組みを採用することにより、これらの情報を容易に変更することが可能となり、様々なケースに応じたシミュレーションが可能となる。

【0137】

以上の説明では、オブジェクト指向を利用しないライブラリの記述例を説明したが、例えば、C++言語等により、ハードウェアモデルライブラリ101、命令セットライブラリ105を、よりスマートに記述しても良い。

【0138】

また、実施の形態1から3に記載の、コンパイラ型のシミュレータ102、インタプリタ型のシミュレータ108等は、「プログラムをコンピュータ読み取り可能に記憶した記録媒体」（CD-ROM、FD、ハードディスク等）に格納して提供できるし、パーソナルコンピュータ、ワークステーションなどにインストールしてから、この装置ごと供給することもできる。

【0139】

ここで、本明細書にいう「プログラムをコンピュータ読み取り可能に記憶した記録媒体」には、複数の記録媒体にプログラムを分散して配布する場合を含む。また、このプログラムが、オペレーティングシステムの一部であるか否かを問わず、種々のプロセスないしスレッド（DLL、OCX、ActiveX等（マイクロソフト社の商標を含む））に機能の一部を肩代わりさせている場合には、肩代わりさせた機能に係る部分が、記録媒体に格納されていない場合も含む。

【0140】

図1には、スタンドアロン形式のシステムを例示したが、サーバー/クライアント形式にしても良い。つまり、1つの端末機のみ、本明細書に出現する全ての要素が含まれている場合の他、1つの端末機がクライアントであり、これが接

続可能なサーバないしネットワーク上に、全部又は一部の要素が実存していても、差し支えない。

【0141】

さらには、図1のほとんどの要素をサーバー側で持ち、クライアント側では、例えば、WWWブラウザだけにしても良い。この場合、各種の情報は、通常サーバ上にあり、基本的にネットワークを経由してクライアントに配布されるものだが、必要な情報が、サーバ上にある時は、そのサーバの記憶装置が、ここにいう「記録媒体」となり、クライアント上にある時は、そのクライアントの記録装置が、ここにいう「記録媒体」となる。

【0142】

さらに、この「プログラム」には、コンパイルされて機械語になったアプリケーションの他、上述のプロセスないしスレッドにより解釈される中間コードとして実存する場合や、少なくともリソースとソースコードとが「記録媒体」上に格納され、これらから機械語のアプリケーションを生成できるコンパイラ及びリンカが「記録媒体」にある場合や、少なくともリソースとソースコードとが「記録媒体」上に格納され、これらから中間コードのアプリケーションを生成できるインタプリタが「記録媒体」にある場合なども含む。

【0143】

【発明の効果】

本発明によれば、ターゲットプロセッサの構成要素単位で機能を、関数等によって、モデル化しておくことにより、コンパイラ型、インタプリタ型の2種類のシミュレータを開発するにあたり、部品を共通化でき、開発効率が向上する。

【0144】

また、アセンブラレベルの最適化も含め、全て高級言語でのソフトウェア開発が可能となる。

【0145】

さらに、翻訳装置を追加することで、コンパイラ型、インタプリタ型の、2種類のシミュレータに用いるソースコードを、一本化できる。

【0146】

また、ライブラリにサイクル数、消費電力ならびにコードサイズを計算する処理を組み込むことにより、コンパイラ型のシミュレータにおいてもパフォーマンス解析が可能になる。

【 0 1 4 7 】

また、これらの情報を容易に変更し、様々なケースに応じたシミュレーションが可能となる。

【図面の簡単な説明】

【図 1】

本発明の実施の形態 1 におけるソフトウェア開発環境のブロック図

【図 2】

同ターゲットプロセッサのブロック図

【図 3】

(a) 同ターゲットプロセッサの命令セットの説明図

(b) 同ターゲットプロセッサのレジスタ指定の説明図

【図 4】

(a) 同演算器のモデル図

(b) 同メモリコントローラのモデル図

【図 5】

同ハードウェアモデルライブラリのヘッダファイルの例示図

【図 6】

同ハードウェアモデルライブラリの実装ファイルの例示図

【図 7】

同命令セットライブラリの実装ファイルの例示図

【図 8】

(a) 同命令セットライブラリのヘッダファイルの例示図

(b) 同ソースコードの例示図

【図 9】

(a) 同インタプリタ型のシミュレータの処理の流れ図

(b) アセンブラによる処理の流れ図

【図 1 0】

同インタプリタ型のシミュレータの実装ファイルの例示図

【図 1 1】

本発明の実施の形態 2 におけるハードウェアモデルライブラリの実装ファイルの例示図

【図 1 2】

同命令セットライブラリの実装ファイルの例示図

【図 1 3】

同ソースコードの例示図

【図 1 4】

本発明の実施の形態 3 における命令セットライブラリの実装ファイルの例示図

【図 1 5】

(a) 同配列の説明図

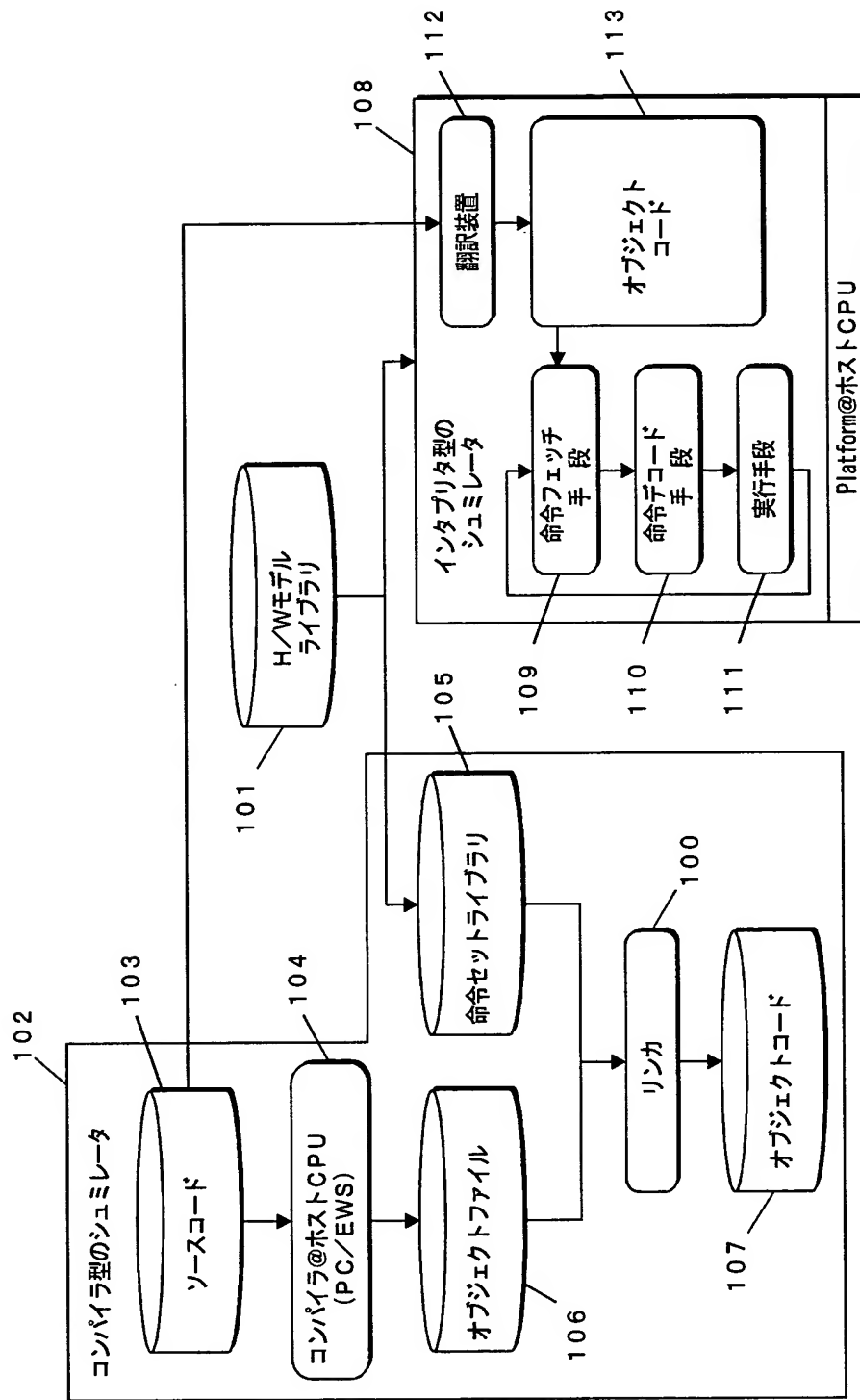
(b) 同ソースコードの例示図

【符号の説明】

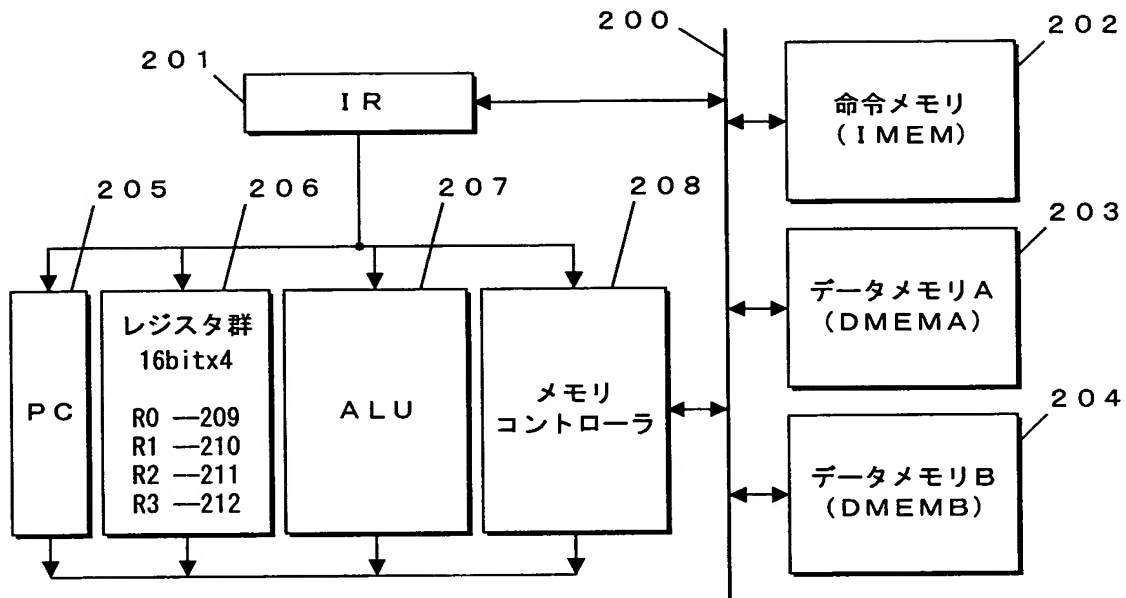
- 1 0 0 リンカ
- 1 0 1 ハードウェアモデルライブラリ
- 1 0 2 コンパイラ型のシミュレータ
- 1 0 3 ソースコード
- 1 0 4 コンパイラ
- 1 0 5 命令セットライブラリ
- 1 0 6 オブジェクトファイル
- 1 0 7 オブジェクトコード
- 1 0 8 インタプリタ型のシミュレータ
- 1 0 9 命令フェッチ手段
- 1 1 0 命令デコード手段
- 1 1 1 実行手段
- 1 1 2 翻訳装置
- 1 1 3 オブジェクトコード

【書類名】 図面

【図 1】



【図 2】



【図 3】

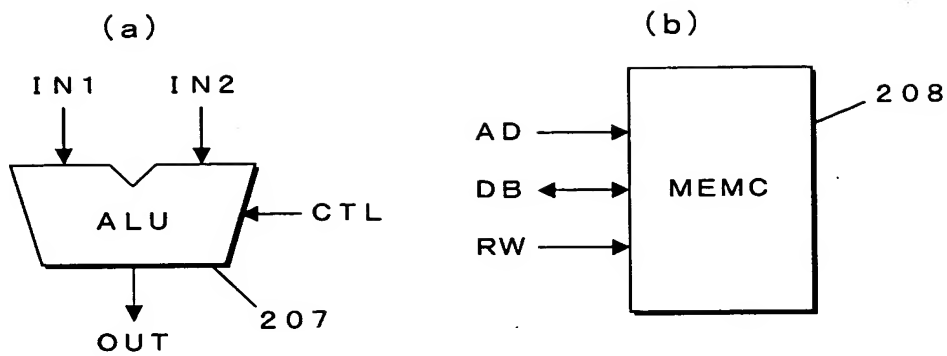
(a)

命令	第1 オペランド	第2 オペランド	機械語
ADD	Ra,	Rb	b000
SUB	Ra,	Rb	b001
AND	Ra,	Rb	b010
OR	Ra,	Rb	b011
LD	Ra,	@Rb	b100
ST	Ra,	@Rb	b101
SET	Ra,	IMD	b110
MOV	Ra,	Rb	b111

(b)

レジスタ	機械語
R0	b00
R1	b01
R2	b10
R3	b11

【図4】



【図5】

```

/* hwmodel.h */

extern short  IMEM[N1];
extern short  DMEMA[N2];
extern short  DMEMA[N3];

extern short  IR;
extern short  PC;
extern short  R0;
extern short  R1;
extern short  R2;
extern short  R3;

extern void ALU(short, short, short*, int);
extern void MEMC(short, short*, int);

enum {
    ADD=0, SUB, AND, OR
};

```

【図 6】

```

short  IMEM[N1];
short  DMEMA[N2];
short  DMEMB[N3];

short  IR, PC, R0, R1, R2, R3;

void ALU(short IN1, short IN2, short *OUT, int CTL) {
    switch (CTL) {
        case 0: // ADD
            *OUT = IN1 + IN2;
            break;
        case 1: // SUB
            *OUT = IN1 - IN2;
            break;
        case 2: // AND
            *OUT = IN1 & IN2;
            break;
        case 3: // OR
            *OUT = IN1 | IN2;
            break;
    }
}

void MEMC(short AD, short *DB, int RW) {
    switch (RW) {
        case 1: // Read
            if (AD < 0x100)
                *DB = DMEMA[AD & 0xFF];
            else
                *DB = DMEMB[AD & 0xFF];
            break;

        case 0: // Write
            if (AD < 0x100)
                DMEMA[AD & 0xFF] = *DB;
            else
                DMEMB[AD & 0xFF] = *DB;
            break;
    }
}

```

【図 7】

```
#include "hwmodel.h"

void ADD(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, ADD);
}

void SUB(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, SUB);
}

void AND(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, AND);
}

void OR(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, OR);
}

void LD(short *RS1, short RS2) {
    MEMC(RS2, RS1, 1);
}

void ST(short *RS1, short RS2) {
    MEMC(RS2, RS1, 0);
}

void SET(short *RS1, short IMD) {
    *RS1 = IMD;
}

void MOV(short *RS1, short RS2) {
    *RS1 = RS2;
}
```

【図 8】

(a)

```

/* inst.h */

extern void  ADD(short *, short);
extern void  SUB(short *, short);
extern void  AND(short *, short);
extern void  OR(short *, short);
extern void  LD(short *, short);
extern void  ST(short *, short);
extern void  SET(short *, short);
extern void  MOV(short *, short);

```

(b)

```

#include "inst.h"

main() {
    .....

    SET(&R0, 0x33);
    SET(&R1, 0x77);

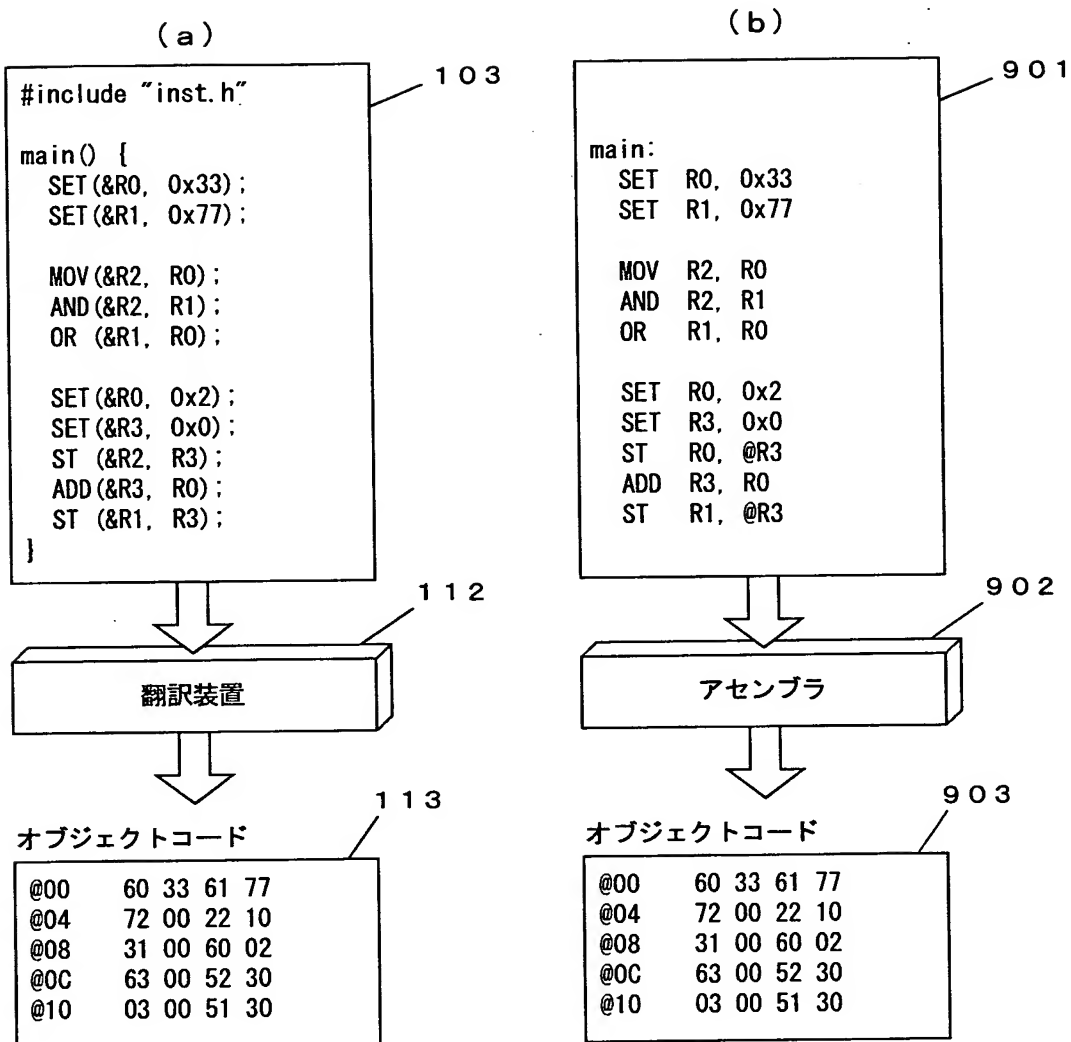
    MOV(&R2, R0);
    AND(&R2, R1);
    OR (&R1, R0);

    SET(&R0, 0x2);
    SET(&R3, 0x0);
    ST (&R2, R3);
    ADD(&R3, R0);
    ST (&R1, R3);
    .....
}

```

1 0 3

【図9】



【図 1 0】

```

#include "hwmodel.h"

enum (
    Fetch=0, Decode, Exec
);

int    cycle;
int    state;
int    *reg[4] = {&R0, &R1, &R2, &R3};

main() {
    cycle = 0;
    state = Fetch;
    while (exec());
    printf("cycle=%d n", cycle);
}

int exec() {
    int cd[3];

    cycle++;

    switch (state) {
        case Fetch:
            IR = IMEM[PC];
            if (IR < 0)
                return 0;
            else {
                PC++;
                state = Decode;
                return 1;
            }
        case Decode:
            cd[0] = (IR>>4)&0x7;
            cd[1] = (IR>>2)&0x3;
            cd[2] = IR & 0x3;
            return 1;
        case Exec:
            if (cd[0]==0x0)
                ALU(*reg[cd[1]],*reg[cd[2]],reg[cd[1]],ADD);
            else if((IR>>4) == 0x1)
                ALU(*reg[cd[1]],*reg[cd[2]],reg[cd[1]],SUB);
            . . . . .
            state = Fetch;
            return 1;
    }
}

```

【図 1 1】

```

short IMEM[N1];
short DMEMA[N2];
short DMEMB[N3];

short IR, PC, R0, R1, R2, R3;
long cycle;
double power;

void ALU(short IN1, short IN2, short *OUT, int CTL) {
    switch (CTL) {
        case ADD:
            *OUT = IN1 + IN2;
            break;
        case SUB:
            *OUT = IN1 - IN2;
            break;
        case AND:
            *OUT = IN1 & IN2;
            break;
        case OR:
            *OUT = IN1 | IN2;
            break;
    }
    cycle += 1;
    power += 0.01;
}

void MEMC(short AD, short *DB, int RW) {
    switch (RW) {
        case 1: // Read
            if (AD < 0x100)
                *DB = DMEMA[AD & 0xFF];
            else
                *DB = DMEMB[AD & 0xFF];
            cycle += 1;
            power += 0.02;
            break;

        case 0: // Write
            if (AD < 0x100)
                DMEMA[AD & 0xFF] = *DB;
            else
                DMEMB[AD & 0xFF] = *DB;
            cycle += 2;
            power += 0.04;
            break;
    }
}

```

【図 1 2】

```
#include "hwmodel.h"

void ADD(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, ADD);
}

void SUB(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, SUB);
}

void AND(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, AND);
}

void OR(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, OR);
}

void LD(short *RS1, short RS2) {
    MEMC(RS2, RS1, 1);
}

void ST(short *RS1, short RS2) {
    MEMC(RS2, RS1, 0);
}

void SET(short *RS1, short IMD) {
    *RS1 = IMD;
    cycle += 1;
    power += 0.005;
}

void MOV(short *RS1, short RS2) {
    *RS1 = RS2;
    cycle += 1;
    power += 0.005;
}
```


【図 13】

```
#include "inst.h"

main( ) {

    .....

    SET(&R0, 0x33);
    SET(&R1, 0x77);

    MOV(&R2, R0);
    AND(&R2, R1);
    OR (&R1, R0);

    SET(&R0, 0x2);
    SET(&R3, 0x0);
    ST (&R2, R3);
    ADD(&R3, R0);
    ST (&R1, R3);

    .....

    printf(cycle="%d\n", cycle);
    printf(power="%f\n", power);
}
```

【図 14】

```
#include "hwmodel.h"

long  cycle;
long  code;
double power;

void ADD(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, ADD);
    cycle += cycle_tb1[0];
    power += power_tb1[0];
    code += 2;
}

void SUB(short *RS1, short RS2) {
    ALU(*RS1, RS2, RS1, SUB);
    cycle += cycle_tb1[1];
    power += power_tb1[1];
    code += 2;
}

.....

void LD(short *RS1, short RS2) {
    MEMC(RS2, RS1, 1);
    cycle += cycle_tb1[4];
    power += power_tb1[4];
    code += 2;
}

void ST(short *RS1, short RS2) {
    MEMC(RS2, RS1, 0);
    cycle += cycle_tb1[5];
    power += power_tb1[5];
    code += 2;
}

.....

void MOV(short *RS1, short RS2) {
    *RS1 = RS2;
    cycle += cycle_tb1[7];
    power += power_tb1[7];
    code += 2;
}
```

【図15】

(a)

命令	index	cycle_tbl[]	power_tbl[]
ADD	0	1	0.01
SUB	1	1	0.01
AND	2	1	0.01
OR	3	1	0.01
LD	4	2	0.02
ST	5	3	0.03
SET	6	1	0.005
MOV	7	1	0.005

(b)

```

long cycle_tbl[8];
long power_tbl[8];

init0 {
    .....
    fp = fopen("table", r);
    for (i=0; i<8; i++)
        fscanf(fp, "%d, %f", &cycle_tbl[i], &power_tbl[i]);
    .....
}

```

【書類名】 要約書

【要約】

【課題】 全て高級言語で開発された1つのソフトウェアを、コンパイラ型、インタプリタ型、シミュレータで、共用できるようにする。

【解決手段】 C言語のソースコードをコンパイルするコンパイラ104と、ターゲットプロセッサのハードウェア構成要素を、C言語でモデル化する関数等と、この関数等を利用して、ターゲットプロセッサの命令に対応してC言語で定義された関数等とを含むライブラリ101、105を備え、ソースコード103は、ライブラリを使用して記述する。

【選択図】 図1

出 願 人 履 歴 情 報

識別番号 [000005821]

1. 変更年月日	1990年 8月28日
[変更理由]	新規登録
住 所	大阪府門真市大字門真1006番地
氏 名	松下電器産業株式会社